Complexity Analysis

Week-02, Lecture-02

Course Code: CSE221
Course Title: Algorithms
Program: B.Sc. in CSE

Course Teacher: Tanzina Afroz Rimi
Designation: Lecturer
Email: tanzinaafroz.cse@diu.edu.bd

STANDARD ANALYSIS TECHNIQUES

- Constant time statements
- Analyzing Loops
- Analyzing Nested Loops
- Analyzing Sequence of Statements
- Analyzing Conditional Statements

CONSTANT TIME STATEMENTS

Simplest case: O(1) time statements

Assignment statements of simple data types

int x = y;

Arithmetic operations:

x = 5 * y + 4 - z;

Array referencing:

A[j] = 5;

Array assignment:

$\forall$ j, A[j] =

5;

Most conditional tests:

if (x < 12) ...

ANALYZING LOOPS[1]

Any loop has two parts:

- How many iterations are performed?
- How many steps per iteration?

```
int sum = 0,j;
for (j=0; j < N; j++)
  sum = sum +j;
```

- Loop executes N times (0..N-1)
- 4 = O(1) steps per iteration

Total time is N * O(1) = O(N*1) = O(N)

ANALYZING LOOPS[2]

What about this **for** loop?

```
int sum =0, j;
for (j=0; j < 100; j++)
   sum = sum +j;
```

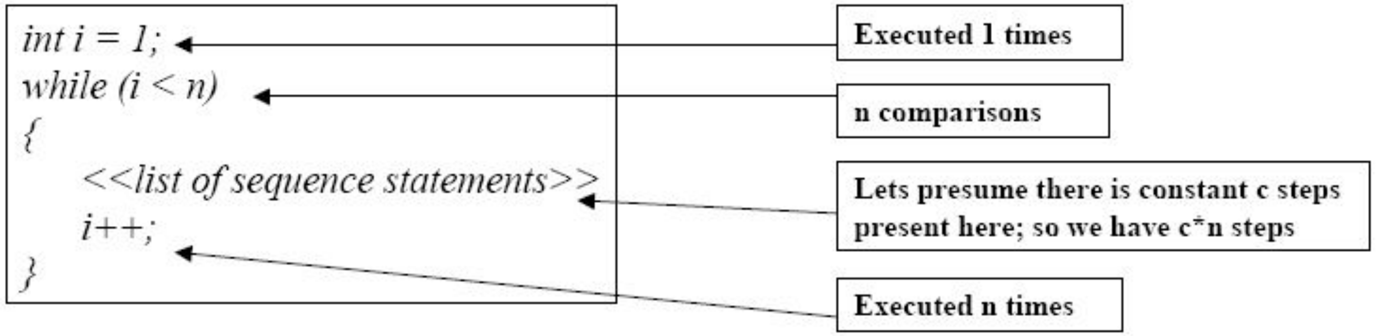
Loop executes 100 times

4 = O(1) steps per iteration

Total time is 100 * O(1) = O(100 * 1) = O(100) = O(1)

ANALYZING LOOPS – LINEAR LOOPS

Example (have a look at this code segment):



Efficiency is proportional to the number of iterations.

Efficiency time function is :

$$
\begin{aligned}
f(n) &= 1 + (n-1) + c*(n-1) +(n-1) \\
&= (c+2)*(n-1) + 1 \\
&= (c+2)n – (c+2) +1
\end{aligned}
$$

Asymptotically, efficiency is : O(n)

ANALYZING NESTED LOOPS[1]

Treat just like a single loop and evaluate each level of nesting as needed:

```
int j,k;
for (j=0; j<N; j++)
  for (k=N; k>0; k--)
    sum += k+j;
```

Start with outer loop:

- How many iterations? N
- How much time per iteration? Need to evaluate inner loop

Inner loop uses O(N) time

Total time is N * O(N) = O(N*N) = O(N)

ANALYZING NESTED LOOPS[2]

What if the number of iterations of one loop depends on the counter of the other?

```
int j,k;
for (j=0; j < N; j++)
   for (k=0; k < j; k++)
      sum += k+j;
```

Analyze inner and outer loop together:

Number of iterations of the inner loop is:

$0 + 1 + 2 + ... + (N-1) = O(N\ )$

HOW DID WE GET THIS ANSWER?

When doing Big-O analysis, we sometimes have to compute a series like: 1 + 2 + 3 + ... + (n-1) + n

i.e. Sum of first n numbers. What is the complexity of this?

Gauss figured out that the sum of the first n numbers is always:

$$\sum_{i=1}^{n} i = \frac{n * (n+1)}{2} = \frac{n^2 + n}{2} = O(n^2)$$

ANALYZING NESTED LOOPS[3]

```
int K=0;
for(int i=0; i<N; i++)
{

   cout <<"Hello";
   for(int j=0; j<K; j--)
          Sum++;

}
```

SEQUENCE OF STATEMENTS

- For a sequence of statements, compute their complexity functions individually and add them up

```
for (j=0; j < N; j++)
    for (k =0; k < j; k++)
        sum = sum + j*k;
for (l=0; l < N; l++)
    sum = sum -l;
System.out.print("sum is now"+sum);
```

$O(N^2)$

$O(N)$

$O(1)$

- Total cost is $O(n^2) + O(n) + O(1) = O(n^2)$

CONDITIONAL STATEMENTS

What about conditional statements such as

```
if (condition)
        statement1;
else
        statement2;
```

where statement1 runs in $O(n)$ time and statement2 runs in $O(n^2)$ time?

We use "worst case" complexity: among all inputs of size n, what is the maximum running time?

The analysis for the example above is $O(n^2)$

DERIVING A RECURRENCE EQUATION

- So far, all algorithms that we have been analyzing have been non recursive

- Example : Recursive power method

```
double power( double x, int n) {
    if ( n == 0)
        return 1.0;                    // base calse
    //else
        return power(x, n-1)*x;        // recursive case
}
```

- If $N = 1$, then running time $T(N)$ is 2

- However if $N \geq 2$, then running time $T(N)$ is the cost of each step taken plus time required to compute power(x,n-1). (i.e. $T(N) = 2+T(N-1)$ for $N \geq 2$)

- How do we solve this? One way is to use the iteration method.

ITERATION METHOD

This is sometimes known as “Back Substituting”.

Involves expanding the recurrence in order to see a pattern.

Solving formula from previous example using the iteration method

Solution : Expand and apply to itself :

Let T(1) = n0 = 2

T(N) = 2 + T(N-1)

= 2 + 2 + T(N-2)

= 2 + 2 + 2 + T(N-3)

= 2 + 2 + 2 + ……+ 2 + T(1)

= 2N + 2 remember that T(1) = n0 = 2 for N = 1

So T(N) = 2N+2 is O(N) for last example.

SUMMARY

Algorithms can be classified according to their complexity => O-Notation

- only relevant for large input sizes

"Measurements" are machine independent

- worst-, average-, best-case analysis

Textbooks & Web References

- Text Book (Chapter 3)
- Reference book iii (Chapter 3)

Thank you

&

Any question?